07

Try with resources

Transcripción

[00:00] ¿Qué tal? Bienvenidos nuevamente. Ya casi estamos a punto de dejar este código mucho más entendible. Vamos a hacer un repaso de cuál fue la línea de razonamiento para llegar a este resultado. Primero, tenemos una conexión que la inicializamos como nula. ¿Por qué? Porque como estamos lanzando la la excepción en el constructor, queremos atrapar esa excepción en el catch.

[00:36] ¿Entonces, qué tenemos que hacer para eso? Crear el objeto adentro del try. Si lo creamos afuera, entonces el catch no va a atrapar a esa excepción cuando suceda. Y después de eso, leemos los datos de la conexión, perfecto, hasta ahí. Y si hay algún error, el catch lo va a atrapar. Y finalmente, yo quiero cerrar la conexión. ¿Pero qué problema tenía yo aquí?

[01:03] Si yo quería cerrar la conexión y había ocurrido algún error, la conexión estaba nula, por lo tanto me iba a dar otro exception aquí del tipo NullPointer. Para asegurarme de no tener un NullPointer en el finally, yo hago una validación que si la conexión es diferente de null, entonces en efecto yo voy a cerrar esa conexión.

[01:24] Si no, entonces no voy a hacer nada. ¿Ahora el código funciona? Sí, el código funciona. ¿El código resuelve el problema? Sí, el código resuelve el problema. Pero el código puede mejorar, y eso es lo que les voy a enseñar aquí, entonces para esto lo que yo voy a hacer es comentar este bloque de código.

[01:46] Voy a usar el tipo de comentario en bloque, que es con slash, un asterisco, y un asterisco y un slash yo delimito el total del bloque que yo quiero comentar, entonces estoy comentando toda esta línea de código. ¿Y qué es lo que voy a hacer ahora? Le voy a presentar a otro nuevo tipo de try. Este es un tipo de try. El try/catch finally.

[02:15] Pero existe otro tipo de try que es algo así: try, abre paréntesis, ya no abre llaves directamente, este abre paréntesis, entonces alguna cosa debe ir adentro de ese paréntesis. Y ahora abre llaves, vamos a dejarlo hasta ahí. Y dentro de los paréntesis normalmente en un método ¿qué ponemos adentro de un paréntesis? Un objeto cierto, que en este caso lo vamos a llamar de recurso.

[02:50] Entonces vamos a ponerle conexión con igual new Conexion. Perfecto. Dénse cuenta lo que yo acabo de hacer, yo he declarado la creación de un objeto del tipo conexión dentro del try. ¿Es válido? Sí, es válido. Pero no está compilando. ¿Y por qué no está compilando? Vamos a ver qué me dice el mensaje.

[03:20] El recurso del tipo conexión, dénse cuenta de ese nuevo nombre que le ha dado, él no me está diciendo el objeto del tipo conexión. Él me dice el recurso. El recurso del tipo conexión no implementa la interfaz AutoCloseable. ¿Qué es la interfaz AutoCloseable? Tranquilos, vamos a descubrirlo dentro de poco.

[03:41] Vamos aquí a la clase conexión. Y como él no implementa la interfaz AutoCloseable y prácticamente me dice que el error es que no la está implementando ¿cómo lo puedo solucionar? Implementando AutoCloseable. Entonces voy a decirle implementes, recuerden su curso de herencia y polimorfismo. AutoCloseable, "Ctrl + espacio", AutoCloseable.

[04:09] Y ahora conexión ya no está compilando. ¿Por qué? Porque necesito implementar los métodos de esta interfaz y esta interfaz tiene un solo método que se llama close, AutoCloseable, método close. Entonces, en el método close,

antes de implementar este método, vamos aquí. Él, en este punto me dice que ahora yo no estoy atrapando la excepción, que puede ser lanzada por AutoClose.

[04:51] Si le doy un throws, entonces, simplemente perfecto, él puede lanzar cualquier excepción en caso el método close lance algún tipo de error cuando está cerrando el recurso, ya sea del tipo NullPointer o alguna cosa así. Para mantener esto simple, por el momento, yo voy a llamar en el método close a mi método cerrar. Perfecto. Entonces, tiene total el sentido.

[05:17] El método close va a llamar al método cerrar conexión. Y ahora mi código, pues compila correctamente, vemos aquí. ¿Y qué voy a hacer ahora si yo ya hice mi new Conexion? Yo ya estoy abriendo aquí mi conexión. Voy a intentar leer los datos, vamos a ver qué es lo que sucede. Voy a guardar aquí y voy a comentar aquí para probar primero en nuestro caso feliz, happy.

[05:58] Voy a ejecutar este código, vemos que él compila correctamente y quiero ver qué pasa. Y miren aquí, cosa curiosa. Abrió la conexión, perfecto, porque él está creando aquí un objeto Conexion new Conexion, recibe los datos, porque porque yo estoy leyendo los datos. ¿Y qué hace él aquí? Cerrando conexión. Esto es lo interesante de esta estructura de try, que se llama try with resource.

[06:27] Try con recursos, es por eso que él ya no lleva el nombre de objeto, sino de recurso. Es un try con recursos. ¿Por qué? Porque en esta estructura try lo que hace es abrir un objeto, inicializa un objeto, instancia un objeto, pero este objeto tiene que implementar la interfaz AutoCloseable, porque esa interfaz te obliga a implementar un método close. ¿Por qué?

[06:59] Porque con ese método close al finalizar, al finalizar esta ejecución, entonces automáticamente, ¿qué va a hacer? Como está llamando a una clase que implementa de AutoCloseable, va allamar al método close y va a cerrar la conexión. Ahora, ese es el primer test. Vamos a ejecutar nuestro segundo test.

[07:25] Supongamos pues que, en efecto, él dio un error aquí al momento de la creación, si yo ejecuto aquí nuevamente vamos a ver, pues que él no cierra la conexión porque simplemente pues dio una excepción. ¿Cómo yo me aseguro de tratar esa excepción? Con catch. Y eso ya lo conocemos. Entonces el try with resources, el try con recursos también me permite usar un catch para atrapar el error.

[07:56] Y yo voy a poner aquí lo mismo. Y voy a decirle aquí "Ejecutando catch". Perfecto. Voy a probar nuevamente y vemos pues que él abrió la conexión y ejecutó el catch, y no llamó pues a al método close, porque nunca se completó este ciclo. Ahora voy a sacarlo de aquí y nuevamente voy a dejarlo en el método leerDatos, voy a guardar aquí. ¿Y ustedes qué creen que es lo que va a pasar? Vamos a ver aquí.

[08:41] Miren qué fue lo que pasó aquí. Él abrió la conexión, recibió los datos, cerró la conexión y ejecutó el catch, en este caso primero fue lanzado, pues el StackTrace porque el error ocurrió primero. En este caso ocurrió después, se van imprimiendo en la consola, de acuerdo a cómo van llegando. La pila de ejecución existe, pero para que los eventos sean registrados en la consola tienen que ser enviados.

[09:17] Entonces como es un procesador, y obviamente pues los eventos no son síncronos uno detrás de otro, en este caso en el caso anterior, primero imprimió el error, pero el el orden correcto es este que acaba de imprimir ahora. Simplemente él abrió la conexión, recibió los datos, cerró la conexión. ¿Por qué? Porque es un AutoCloseable y ejecutó el catch, y en el catch me imprimió el StackTrace.

[09:45] Pero miren, hace exactamente lo mismo que todo este bloque de código de try/catch finally. Yo lo he resumido en tan solo seis líneas. Miren la diferencia. Es así cómo, para este tipo de casos, en el caso de la conexión, tiene total sentido usar esta estructura try with resource. Este tipo de try de aquí también viene mucho en examen de certificación. Entonces, para los que ya

están pensando, pues en tomar un examen es totalmente válido, estudiar esta parte de aquí.

[10:29] ¿Y en qué casos me conviene usar este tipo de try con finally y en qué casos me conviene usar este de aquí? Quién va a decidir eso es esta interfaz de aquí. Si tú necesitas que este objeto haga algo al momento de dejar de existir, en este caso, cuando se ha cerrado, cuando se ha cerrado me refiero cuando ya no voy a utilizarlo más, que ya se convierte en un recurso, un recurso es algo que yo agarro, utilizo y después desecho.

[11:05] Si fuera un recurso, yo necesito cerrarlo, después puedo implementar la interfaz, AutoCloseable y automáticamente él va a cerrar ese recurso y va a ejecutar todo normalmente, pero si mi objeto no implementa AutoCloseable, yo tengo esta otra estructura de aquí para manejar mis errores incluso en distribuciones de Java antes de Java 1.7, si no me equivoco, esto de aquí no existía y la única estructura para tratamiento de errores válida era esta de aquí.

[11:42] O sea, este sería mi código optimizado al máximo que yo hubiera podido escribir si estuviéramos usando Java hasta la versión 1.6. Pero a partir de 1.7 llega esta estructura try with resource, que simplifica mucho, mucho el tratamiento de errores y ejecuciones de finally cuando mi recurso implementa AutoCloseable y yo necesitaría cerrar alguna conexión o cerrar algún lector de datos, alguna cosa de ese tipo.

[12:14] Entonces, nuevamente, practiquen mucho esta estructura de try, el try with resource, para mí fue un poco más complicado entender esta sintaxis del try que esta de aquí, pero una vez que agarran la práctica de cómo funciona la declaración de un recurso adentro del try y saber que el método close va a ser llamado de cualquier forma, después se les va a hacer bien fácil.

[12:44] Y recuerden, aquí no llamamos de objetos, llamamos de recursos. ¿Por qué? Porque el recurso es algo que al final yo voy a tener que cerrar, sea una conexión, un lector de datos, alguna cosa así, yo tengo que cerrarlo. Ahí, try

with resource es lo que nos va a salvar el día, si no fuera así, si tuviera un objeto normal, una estructura de try tradicional, nos va a servir muy bien.

[13:09] Recordando aquí nuevamente, voy a descomentar aquí. Recordando aquí. ¿Try puede vivir solo? No. Try no puede vivir solo, no va a compilar. Try necesita de un catch o de un finally y puede tener ambos. El finally es opcional. Si nosotros queremos ejecutar un código, haya o no haya error. Y no puede ir el finally antes del catch. Obviamente. Bueno.

[13:49] Esta fue la nueva estructura de try que quería presentarles. Ya sabemos en qué casos es válido usar esta estructura de aquí, en qué casos nos conviene mucho usar esta estructura de aquí. Les recomiendo mucha práctica y nos vemos.